ASSIGNMENT NO.3

Title : Implementing Feed forward neural networks

Aim: Implementing Feedforward neural networks with Keras and TensorFlow

Import the necessary packages

Load the training and testing data (MNIST/CIFAR10)

Define the network architecture using Keras

Train the model using SGD

Evaluate the network

Plot the training loss and accuracy

Theory :

Feedforward Neural Network:

A feedforward neural network is a type of artificial neural network in which nodes' connections do not form a loop.

- Often referred to as a multi-layered network of neurons, feedforward neural networks are so named because all information flows in a forward manner only.
- The data enters the input nodes, travels through the hidden layers, and eventually exits the output nodes.
- No links exist in the network that could get used to by sending information back from the output node.

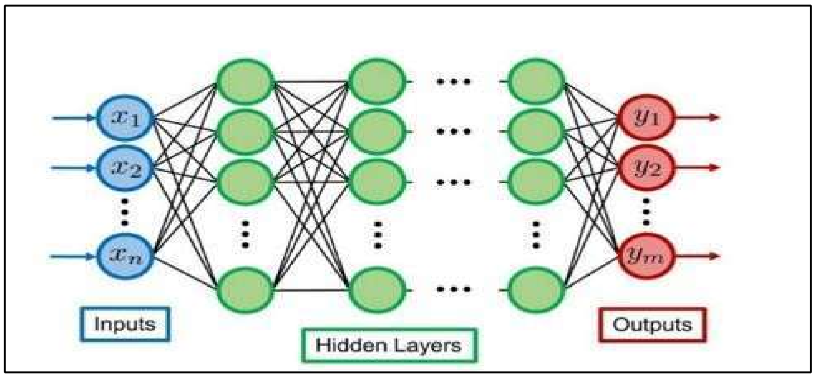


Figure 1: Feed Forward Neural Network

Feedforward neural networks are made up of the following:

- Input layer: This layer consists of the neurons that receive inputs and pass them on to the other layers. The number of neurons in the input layer should be equal to the attributes or features in the dataset.
- Output layer: The output layer is the predicted feature and depends on the type of model you're building.
- Hidden layer: In between the input and output layer, there are hidden layers based on the type of model. Hidden layers contain a vast number of neurons which apply transformations to the inputs before passing them. As the network is trained, the weights are updated to be more predictive.
- Neuron weights: Weights refer to the strength or amplitude of a connection between two neurons. If you are familiar with linear regression, you can compare weights on inputs like coefficients. Weights are often initialized to small random values, such as values in the range 0 to 1.

Working of Feed Forward Neural Network:

A Feed Forward Neural Network is commonly seen in its simplest form as a single layer perceptron.

In this model, a series of inputs enter the layer and are multiplied by the weights. Each value is then added together to get a sum of the weighted input values. If the sum of the values is above a specific threshold, usually set at zero, the value produced is often 1, whereas if the sum falls below the threshold, the output value is -1.

Applications of Feed Forward Neural Networks:

1. Facial Recognition
2. Social Media
3. Aerospace Engineering
4. Healthcare

The cost function of a neural network will be the sum of errors in each layer. This is done by finding the error at each layer first and then summing the individual error to get the total error.

The Mean Squared Error measures how close a regression line is to a set of data points.. Mean square error is calculated by taking the average, specifically the mean, of errors squared from data as it relates to a function.

The loss function in the neural network is meant for determining if there is any correction the learning process needs. The output layer neurons will be equal to the number of classes. To compare the difference between predicted and true probability distribution.

Cross-Entropy loss is a most important cost function. It is used to optimize classification models.

It is a metric used in machine learning to measure how well a classification model performs. The loss (or error) is measured as a number between 0 and 1, with 0 being a perfect model. The goal is generally to get your model as close to 0 as possible.

Kernels (weights) — used to scale input and hidden node values. Each connection typically holds a different weight. Biases — used to adjust scaled values before passing them through an activation function.

In general, a kernel is a positive-semidefinite symmetric function of two inputs which represents some notion of similarity between the two inputs.

MNIST and CIFAR-10 are both widely used datasets for image recognition tasks, but they have some differences that can make one more suitable for a particular task than the other. Here are some of the pros and cons of each dataset: MNIST: Pros:MNIST is a relatively small dataset, with only 70,000 images.

Flattening is converting the data into a 1-dimensional array for inputting it to the next layer. We flatten the output of the convolutional layers to create a single long feature vector. And it is connected to the final classification model, which is called a fully-connected layer.

Difference between Sigmoid and Softmax activation function:

Softmax is used for multi-classification in the Logistic Regression model, whereas Sigmoid is used for binary classification in the Logistic Regression model.

Main reason why the Softmax is cool. It makes sure that the sum of all our output probabilities is equal to one.

The Sigmoid and Softmax Activation Functions produce different results.

$$f(x) = sigmoid(x) = \frac{1}{1 + e^{-x}}$$

Sigmoid input values: -0.5, 1.2, -0.1, 2.4

Sigmoid output values: 0.37, 0.77, 0.48, 0.91

$$\text{softmax}(z_j) = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}} \text{ for } j = 1,\ldots,K$$

SoftMax input values: -0.5, 1.2, -0.1, 2.4

SoftMax output values: 0.04, 0.21, 0.05, 0.70

An optimizer is an algorithm or function that adapts the neural network's attributes, like learning rate and weights. Hence, it assists in improving the accuracy and reduces the total loss. But it is a daunting task to choose the appropriate weights for the model.
An epoch is a single iteration though the training data. Each sample from your training dataset will be used once per epoch, whether it is for training or validation. Therefore, the more epochs, the more the model is trained.

Steps/ Algorithm

Dataset link and libraries :

Dataset : MNIST or CIFAR 10 : kaggel.com

You can download dataset from above mentioned website. Libraries required :

Pandas and Numpy for data manipulation Tensorflow/Keras for Neural Networks

Scikit-learn library for splitting the data into train-test samples, and for some basic model evaluation implementing-feedforward-neural-networks-withker

$$f(x) = sigmoid(x) = \frac{1}{1 + e^{-x}}$$

Import following arizer (sklearn.preprocessing) ii) classification_report (sklearn.metrics) .

Import Following libraries from tensorflow.keras : models , layers,optimizers,datasets ,baclend and set to respective values.

Grab the MNIST dataset or required dataset.

Flatten the dataset.

If required do the normalization of data .

Convert the labels from integers to vectors.(specially for one hot coding)

Decide the Neural Network Architecture : i) Select model (Sequential recommended)
ii) Activation function (sigmoid recommended) iii) Select the input shape iv) see the weights in the output layer

Train the model\: i) Select optimizer (SGD recommended) ii) use model that .fit to start training

ii) Set Epochs and batch size

Call model.predict for class prediction.

Plot training and loss accuracy

Calculate Precision, Recall, F1-score, Support Repeat

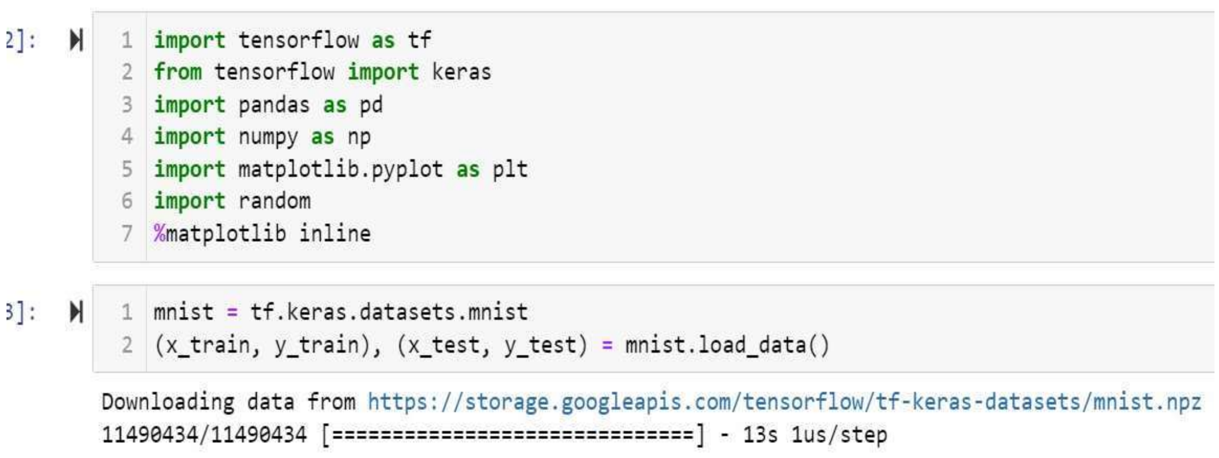
for CIFAR dataset.

Input Code:

```
import tensorflow as tf from
tensorflow import keras
import pandas as pd import
numpy as np import
matplotlib.pyplot as plt import
random %matplotlib inline

mnist = tf.keras.datasets.mnist
(x_train, y_train), (x_test, y_test) = mnist.load_data()

plt.matshow(x_train[1]) plt.imshow(-x_train[0],
cmap=”gray”)

x_train = x_train / 255 x_test
= x_test / 255

model = keras.Sequential([ keras.layers.Flatten(input_shape=(28,
28)), keras.layers.Dense(128, activation=”relu”),
keras.layers.Dense(10, activation=”softmax”)
])

model.summary()

model.compile(optimizer=”sgd”,
loss=”sparse_categorical_crossentropy”, metrics=[‘accuracy’])

history=model.fit(x_train, y_train,validation_data=(x_test,y_test),epochs=10)
test_loss,test_acc=model.evaluate(x_test,y_test) print(“Loss=%.3f”
%test_loss) print(“Accuracy=%.3f” %test_acc)

n=random.randint(0,9999)
plt.imshow(x_test[n]) plt.show()
```

```
x_train x_test

predicted_value=model.predict(x_test)
plt.imshow(x_test[n]) plt.show()

print(predicted_value[n])

# history.history() history.history.keys()
# dict_keys([‘loss’, ‘accuracy’, ‘val_loss’, ‘val_accuracy’])

plt.plot(history.history[‘accuracy’])
plt.plot(history.history[‘val_accuracy’])
plt.title(‘model accuracy’)
plt.ylabel(‘accuracy’) plt.xlabel(‘epoch’)
plt.legend([‘Train’, ‘Validation’], loc=’upper left’) plt.show()

# history.history() history.history.keys()
# dict_keys([‘loss’, ‘accuracy’, ‘val_loss’, ‘val_accuracy’]) plt.plot(history.history[‘loss’])
plt.plot(history.history[‘val_loss’]) plt.title(‘model loss’) plt.ylabel(‘loss’) plt.xlabel(‘epoch’)
plt.legend([‘Train’, ‘Validation’], loc=’upper left’) plt.show()
```

Output:



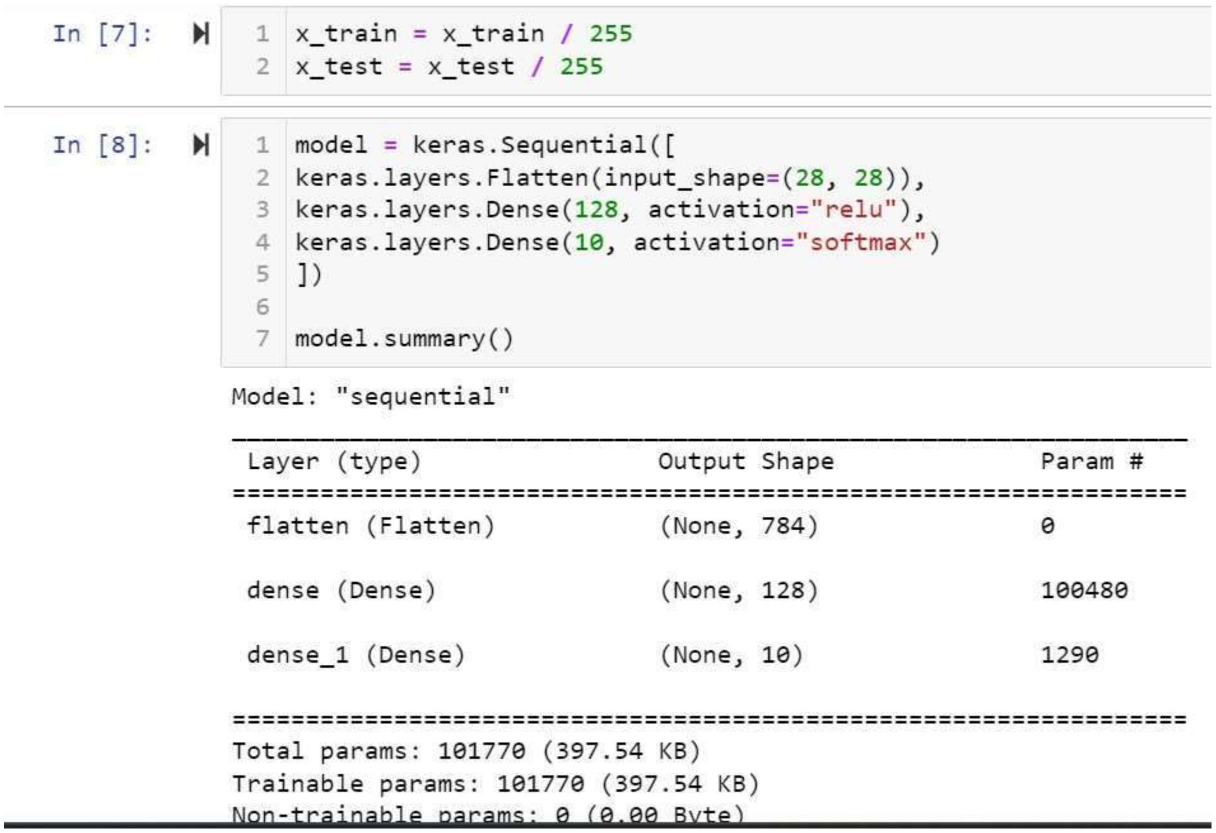
```
2]:
import tensorflow as tf
from tensorflow import keras
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import random
%matplotlib inline
```

```
3]:
mnist = tf.keras.datasets.mnist
(x_train, y_train), (x_test, y_test) = mnist.load_data()
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz
11490434/11490434 [==============================] - 13s 1us/step
```

Figure 2: Output 1

The above figure shows the importing of libraries and dataset.



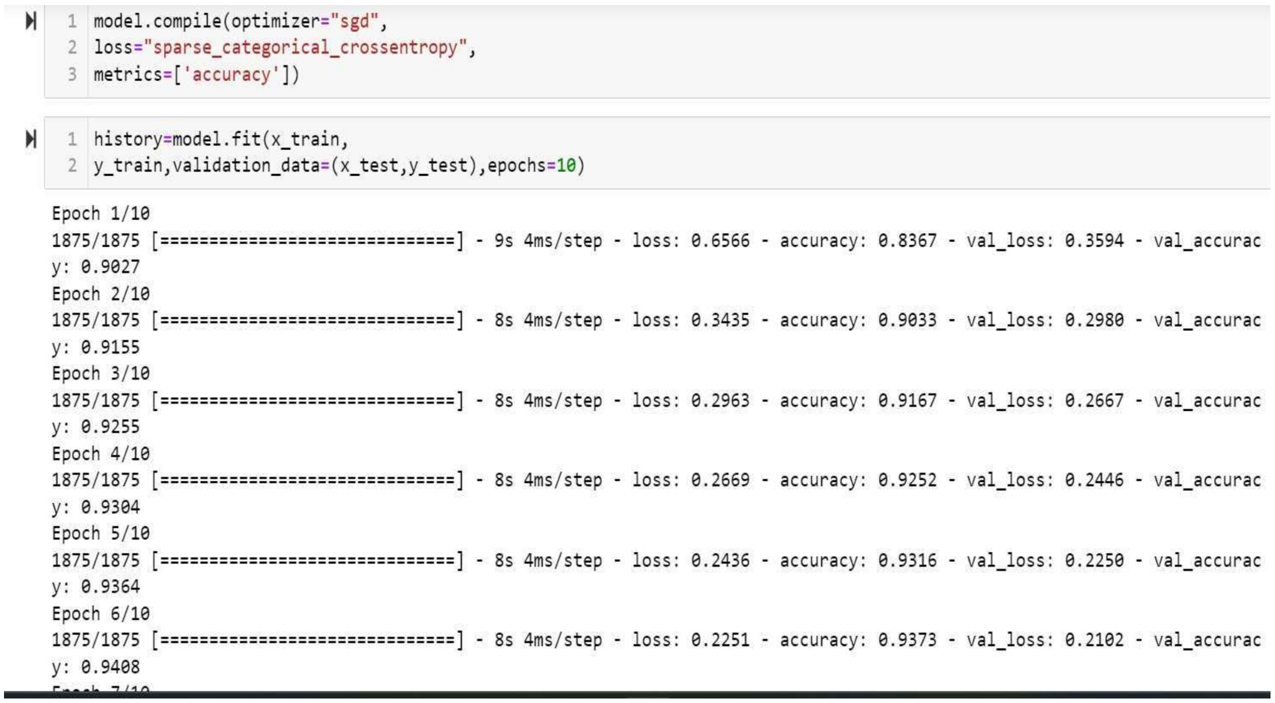
```
In [7]:
x_train = x_train / 255
x_test = x_test / 255
```

```
In [8]:
model = keras.Sequential([
keras.layers.Flatten(input_shape=(28, 28)),
keras.layers.Dense(128, activation="relu"),
keras.layers.Dense(10, activation="softmax")
])

model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 flatten (Flatten)           (None, 784)               0

 dense (Dense)               (None, 128)               100480

 dense_1 (Dense)             (None, 10)                1290

=================================================================
Total params: 101770 (397.54 KB)
Trainable params: 101770 (397.54 KB)
Non-trainable params: 0 (0.00 Byte)
```

Figure 3: Output 2

The above figure shows the training and testing data (MNIST/CIFAR10).



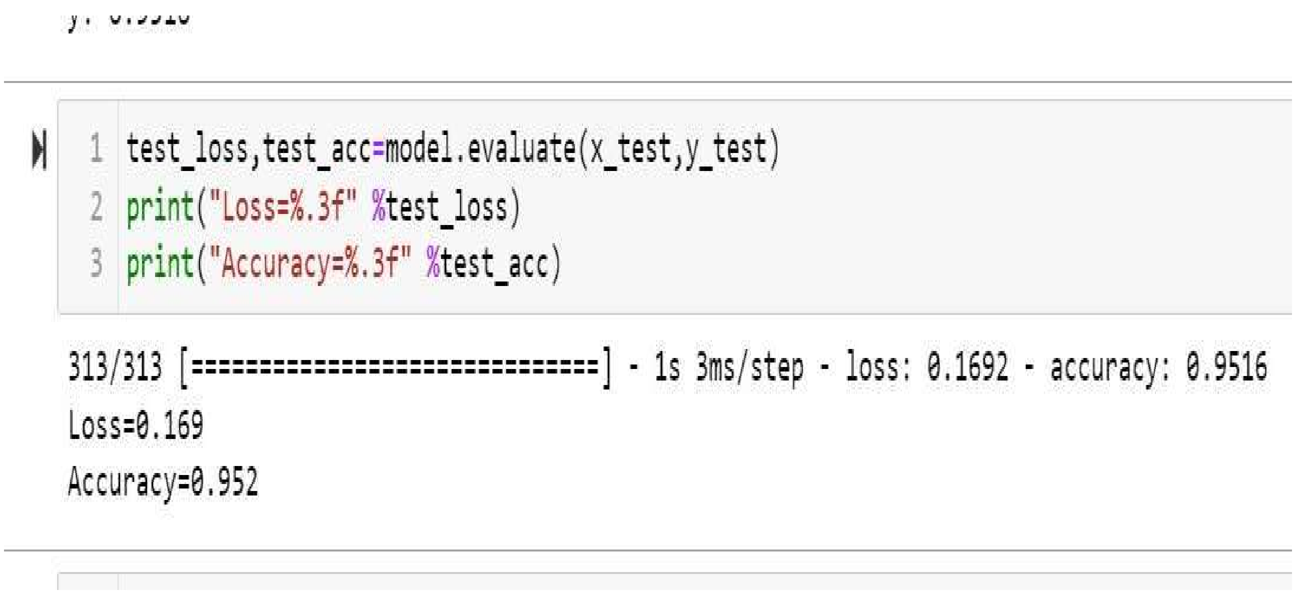
```
model.compile(optimizer="sgd",
loss="sparse_categorical_crossentropy",
metrics=['accuracy'])
```

```
history=model.fit(x_train,
y_train,validation_data=(x_test,y_test),epochs=10)
```

```
Epoch 1/10
1875/1875 [==============================] - 9s 4ms/step - loss: 0.6566 - accuracy: 0.8367 - val_loss: 0.3594 - val_accurac
y: 0.9027
Epoch 2/10
1875/1875 [==============================] - 8s 4ms/step - loss: 0.3435 - accuracy: 0.9033 - val_loss: 0.2980 - val_accurac
y: 0.9155
Epoch 3/10
1875/1875 [==============================] - 8s 4ms/step - loss: 0.2963 - accuracy: 0.9167 - val_loss: 0.2667 - val_accurac
y: 0.9255
Epoch 4/10
1875/1875 [==============================] - 8s 4ms/step - loss: 0.2669 - accuracy: 0.9252 - val_loss: 0.2446 - val_accurac
y: 0.9304
Epoch 5/10
1875/1875 [==============================] - 8s 4ms/step - loss: 0.2436 - accuracy: 0.9316 - val_loss: 0.2250 - val_accurac
y: 0.9364
Epoch 6/10
1875/1875 [==============================] - 8s 4ms/step - loss: 0.2251 - accuracy: 0.9373 - val_loss: 0.2102 - val_accurac
y: 0.9408
```

Figure 4: Output 3

The above figure shows training of the dataset with 10 epochs.



```
test_loss,test_acc=model.evaluate(x_test,y_test)
print("Loss=%.3f" %test_loss)
print("Accuracy=%.3f" %test_acc)
```

```
313/313 [==============================] - 1s 3ms/step - loss: 0.1692 - accuracy: 0.9516
Loss=0.169
Accuracy=0.952
```

Figure 5: Output 4

The above figure shows Loss and Accuracy of our model after training and testing.

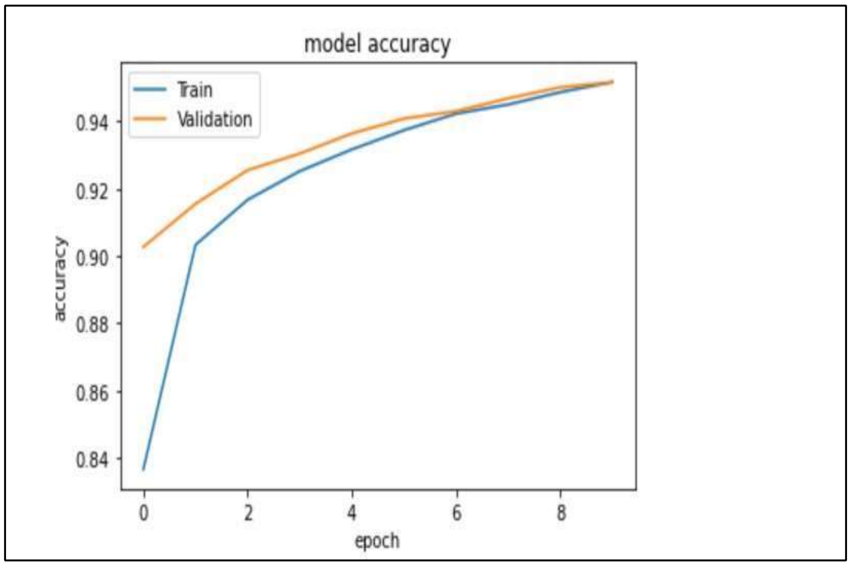


Figure 5: Graph for Accuracy

The above figure shows the graph of Accuracy of our model using matplotlib library.

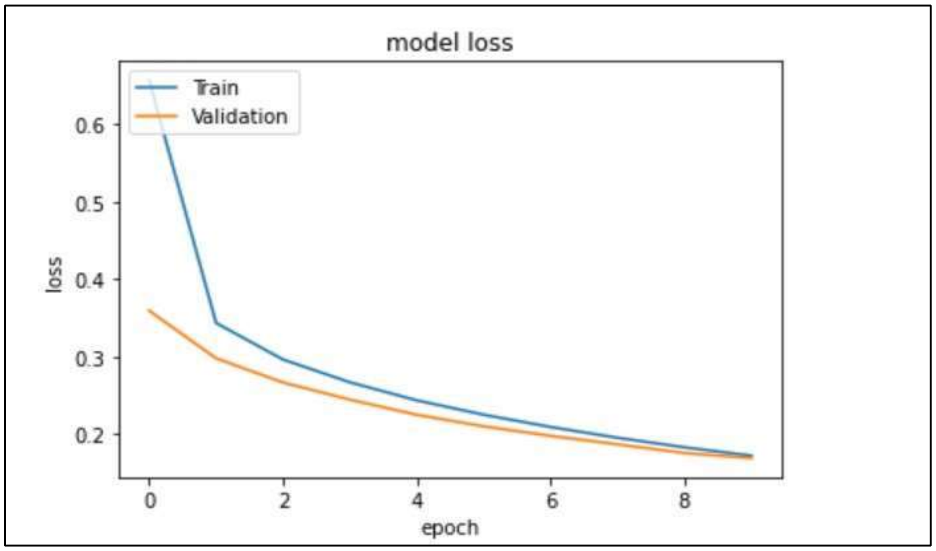


Figure 6: Graph for Loss

The above figure shows the graph of Loss of our model using matplotlib library.

Conclusion:

Thus, FFNN deep learning algorithm is implemented and tested for MNIST handwritten digit recognition.

After training for 10epoch, the accuracy of the model for digit recognition is 95.2% and loss 16.9%.